Consistent and Efficient Clouds
NSF "The Science of Cloud Computing" Submission

Robbert van Renesse (rvr@cs.cornell.edu)

Department of Computer Science, Cornell University

Research Topic 1: Fault Masking in the Cloud
Research Topic 2: Network Support for the Cloud

My research interest is in fault tolerance in the cloud. This comprises trustworthiness of storage, computing, and networking, and can involve multiple datacenters. The failure models I have considered include crash and Byzantine host failures as well as network partitioning. The consistency models I have considered range from weak "eventual consistency" to ACID-style consistency.

The work that I have done that has probably had the most impact is the Astrolabe system (ACM TOCS, 2003). This DHT-like system was acquired by Amazon.com and deployed on tens of thousands of nodes spanning multiple continents. A peer-to-peer system, it collects data from partitipating nodes, aggregates the data as specified by a SQL aggregation query, and uses the result to route requests to servers. The gossip-based design also heavily influenced the design of the Amazon.com Dynamo and S3 services, and the popular Facebook Cassandra storage service.

Academically successful in terms of publications, but without industry adoption, has been my work on Byzantine fault tolerance. Several studies have shown that only a minority of failures are crash failures. It would seem then that the cloud computing industry would be eager to deploy Byzantine techniques. However, such techniques are considered overly expensive, complex, and non-scalable for use in cloud computing. My students and I have addressed the complexity and scalability problem, by demonstrating that scalable crash tolerant systems can be converted automatically into scalable systems that tolerate Byzantine faults (Usenix NSDI, 2008). But cost is still an issue.

A third research direction has been to pursue formal methods in the design and implementation of practical systems. Formal methods help with analyzing the assumptions that a system makes, and exposes thereby its vulnerabilities.

Cloud computing has its roots in services where fast response is important for revenue, but where consistency guarantees can be relaxed. If Amazon sends two copies of a book instead of the one that a customer ordered, this is no big disaster. But as cloud computing gets embraced for health services, financial services, and military operations, not only is fast response even more critical, but lack of consistency can also lead to loss of life. At the same time, it is likely that operators will demand low cost, and low operational cost is also important to the environment. **What we desperately need are scalable systems that provide highly available and strongly consistent (serializable ACID) systems that can tolerate a large class of failures and have low cost of ownership at the same time.**

In highly publicized articles in ACM Queue 2008 ("Eventually Consistent," Werner Vogels, Amazon.com and "BASE, an ACID alternative," Dan Pritchett, eBay), the cloud computing industry points to the FLP impossibility result that says that in an asynchronous environment it is impossible to guarantee termination of a consensus protocol in the presence of faults. However, asynchrony is a total lack of timing assumptions, while in practice one can assume bounds on clock drift and the time that a message is sent and processed. Even if such bounds are unknown, one can design a consensus protocol that is guaranteed to terminate. Those articles also point to Brewer's now famous CAP hypothesis that states that among Consistency, Availability, and Partition-Tolerance, one can implement at most two. Again, this is rooted in the absence of timing assumptions, and it simply does not apply in environment that provides (unknown) bounds on network latencies, processing latencies and clock drift. Finally, the industry points to lower bounds on consensus protocols used in systems that guarantee consistency—the high cost associated with ACID guarantees—particularly if Byzantine failures are to be tolerated. But in many cases, such costs can be effectively amortized.

FLP, CAP, and lower bound results, they are all very important to our understanding of the limitations of distributed systems. However, I believe that all hope is not lost, and that if we act quickly, we can develop techniques that will lead to affordable but highly dependable cloud services. The answer is in examining the assumptions that one can make in building such services.

For example, in (partially) asynchronous Byzantine consensus, we usually assume that there is a bound f on the number of nodes that can experience a failure, and we assume that those nodes cannot break cryptographic building blocks. Under such assumptions, it is well-known that one needs more than $3f$ nodes to maintain consistency. But the industy demands that at most $f + 1$ nodes be used. We can reach this by combining two means. First, we need to examine what additional assumptions we can make about Byzantine failures and network connectivity *in practice*. Second, we can amortize the cost of reconfiguration at the time of a suspected failure using techniques such as used in Vertical Paxos (Leslie Lamport et al., 2009). Third, we need to be able to regenerate replicas on-the-fly. I believe this can be done by the new replica demand paging its state from existing replicas.

We need fully serializable ACID guarantees across large storage systems. Such systems tend to be partitioned and include replication, and providing ACID guarantees at low cost is not an easy matter. However, there is no result that states that doing so is impossible. Again, assumptions need to be examined, and for efficiency different techniques may need to be used for different workloads.

Besides storage systems, there will be a need for computing services that continually process incoming event streams. Yet, those services need to tolerate failures, as well as need hardware and software upgrades. Current such systems most often run off-line if they are to produce accurate results, or provide only weak consistency guarantees and approximate results. I am convinced that it is possible to build usable reconfigurable stream processing systems that provide consistency, if we carefully examine what assumptions we can make about the environment and the workloads.

Finally, we should not forget the demands such systems place on networks. With demand for information and multi-media data rapidly increasing, we need to examine what are efficient means for distribution. This requires a careful examination of the data itself, and look into efficient and robust routing, redundancy in the network itself, deduplication, and so on, both for intra-datacenter networking and the Internet itself.